Internet of Things and Smart Cities Ph.D. School 2013 -
University of Parma

Hands on Contiki OS and Cooja Simulator: Exercises (Part II)

Ing. Pietro Gonizzi (pietro.gonizzi@studenti.unipr.it)
Dr. Simon Duquennoy (simonduq@sics.se)

17-09-2013

1 Introduction

Welcome to this Contiki programming course! Contiki is a state-of-the-art, open source operating system for sensor networks and other networked embedded devices [1].
You will look at various cool Contiki features such as the uIP stack (IPv6, RPL, 6LoWPAN), HTTP/CoAP, power profiling and others. We will use Tmote Sky boards as our hardware target. We will also use the Cooja network simulator which emulates Tmote Sky nodes and connects them.
You can use the Instant Contiki development environment in the exercises. Instant Contiki is a single-file download that contains the Contiki source code and all necessary compilers and tools required for developing software for Contiki. The Instant Contiki environment is a Ubuntu Linux installation that runs within the VMware Player virtual machine execution environment. VMware Player is available for free at the VMware website. If you have Contiki working on your native machine without Instant Contiki that is of course also fine.

2 Getting Started

Before starting with the actual exercises, make sure that your development setup works by conducting the steps below.

2.1 Start Instant Contiki

Open the file

`instant-contiki.vmx`

to start VMware and Instant Contiki.

2.2 Log In

When the login screen appears, log in to Instant Contiki:

- Username: **user**
- Password: **user**

2.3 Open a Terminal Window

After logging in, click on the terminal icon to start a terminal window. You may also need to adjust the language and keyboard settings of your system.

2.4 Verify that your system works

In the terminal window, go to the hello world example directory, and compile for the native platform (this means Contiki is compiled to linux process rather than to a ready-to-burn image for sensor nodes):

```
$ cd contiki
$ cd examples/hello-world
$ make TARGET=native
```

Wait for the compilation to finish. Run the Hello World program in Contiki:

```
$ ./hello-world.native
```

The program should print the words "Hello, world" on the screen and then appear to hang. In reality, Contiki is still running correctly, but will not produce any more output because the Hello World program has finished. Press Ctrl+C on the keyboard to quit.
Note that Contiki uses `printf` to issue logs and messages. This can also be used when you have a sensor node that is connected via a serial line, e.g. USB, to a laptop.

2.5 Run Hello World on a Tmote Sky Node

Now, run the hello-world example on real hardware. In this course, we use the Tmote Sky, which features a 16-bit msp430 MCU, 10 kB RAM, 48 kB ROM, a cc2420 802.15.4 radio transceiver, an external Flash memory, and temperature, humidity and brightness sensor.

2.5.1 Compile Contiki for the Tmote Sky

Compile Contiki for the `sky` platform:

```
$ make hello-world.upload TARGET=sky
```

This triggers the cross-compilation of Contiki using msp430-gcc and including the drivers that are specific to the `sky` platform. The outcome is the ELF file `hello-world.sky`, a ready-to-burn firmware. Playing with `objdump`, `nm`, and `size` is a good idea to get familiar with what a Contiki firmware look like.

Exercise 1. Use `size` to find out how much RAM and ROM the firmware needs to run. Do you have any comments/thoughts about runtime stack usage?

□

2.5.2 Connect the Tmote Sky

As we are going to use the `sky` target a few more times, ask Contiki to remember it by using:

```
$ make TARGET=sky savetarget
```

Put a Tmote Sky in a computer's USB port. The Tmote Sky will appear in the top of the Instant Contiki (VMware Player) window with the name "Future Device" in the menu "Player/Removable Devices". Click on the name to connect the Tmote Sky to Instant Contiki (note this will disconnect the device from the host OS). Run the following to find all connected sky nodes and their associated tty file:

```
$ make sky-motelist
```

2.5.3 Upload the Firmware

Upload the firmware to your node (the argument `MOTE` is optional; when unspecified, all nodes are reprogrammed):

```
$ make hello-world.upload MOTE=1
```

During the uploading the Tmote Sky should quickly flash the red LEDs next to the USB connector.
Once the upload is terminated, connect to the USB port to view the program output:

```
$ make login MOTE=1
```

Press the reset button on the Tmote Sky (the one further away from the USB connector) and you should get some Contiki startup log messages followed by `Hello, world`.

Exercise 2. Install the `test-button` program from the `examples/sky` directory. Connect to the USB port (`make login`) and press the non-reset button (the one closest to the USB connector).

□

2.6 Run Hello World in Cooja

Cooja is a network simulator that is able to emulate Tmote Sky (and other) nodes. The code executed by the node is the exact same firmware you may upload to physical nodes.
Start Cooja by using the desktop icon or running `ant run` from the directory `tools/Cooja`. Now that Cooja is up and running, you can try it out with an example simulation. Click the *File* menu and click *New simulation*. Cooja now opens up the *Create new simulation* dialog. In this dialog, you may choose to give your simulation a new name, e.g., *Hello-world*.
After clicking the *Create* button, Cooja brings up the new simulation. The **Network** window, at the top left of the screen, shows all the motes in the simulated network - it is empty now, since you have no motes in your simulation. The **Timeline** window, at the bottom of the screen, shows all communication events in the simulation over time - very handy for understanding what goes on in the network. The **Mote output** window, on the right side of the screen, shows all serial port printouts from all the motes. The **Notes** window on the top right is where you can put notes for your simulation. And the **Simulation control** window is where you start, pause, and reload your simulation.
Before you can simulate your network, you must add one or more motes. You do this via the *Motes* menu, where you click on *Add motes*. Since this is the first mote you add, you must first create a mote type to add. Click *Create new mote type* and select one of the available mote types. In this course, you click *Sky mote* to create an emulated Tmote Sky mote type.
Cooja opens the *Create Mote Type* dialog, in which you can choose a name for your mote type as well as the Contiki application that your mote type will run. Give a name to your mote type and click on the *Browse* button on the right hand side to choose your Contiki application. Go to the directory `/home/user/contiki/examples/hello-world` and select the `hello-world.c` source file. Now Cooja will verify that the selected Contiki application compiles for the platform that we have selected. Click the *Compile* button. The compilation output will show up in the white panel at the bottom of the window. If the compilation is fine, click the *Create* button. You will be asked to enter the number of motes of your simulation. Once you are done, click the *Start* button in the *Simulation control* window to start the simulation.

2.6.1 Save, Load and Reload in Cooja

Cooja allows for saving and loading simulation configurations. When a simulation is saved, all active plugins are also stored with the configuration. The

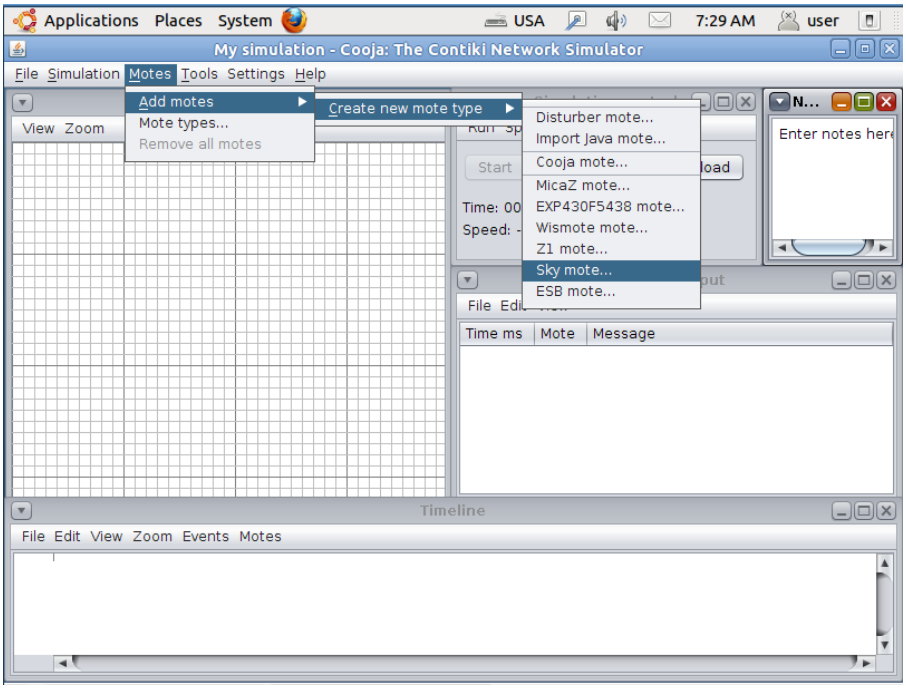

Figure 2.1: The Cooja simulator

state of a current simulation is however not saved; all nodes are reset when the simulation is loaded again. To save your current simulation:

- Click menu item: **File, Save simulation**.

Simulations are stored with the file extensions ".csc". To later load a simulation:

- Click menu item: **File, Open simulation, Browse...**. Select a simulation configuration.

When a simulation is loaded, all simulated Contiki applications are recompiled. A functionality similar to saving and loading simulations, is *reloading a simulation*. Reloading can be used to reset the simulation – to restart all nodes. More importantly, reloading a simulation will recompile all Contiki code, useful while developing Contiki programs. To reload your current simulation:

- Click menu item: **File, Reload simulation**, or press **Ctrl+R**

2.6.2 Radio model in Cooja

Once you have added nodes, you may want to check if they can communicate with each other. From the **Network** window, left-click on the *View* textbox

and tick *Radio Environment (UDGM)* in the text window that pops up. If you now left-click on one of the nodes, you will see a green circle around the selected node. The green circle presents the **transmission range** of the node, i.e. the selected node can communicate with all other nodes within that circle. You may also see a grey circle around the green circle. This circle represents the **interference range**. That is when the selected nodes transmits packets, a node in the grey area cannot receive packets correctly but it is interferred which means that it is not able to receive packets from other nodes when the selected node communicates simultaneously. The areas that are not covered by the green or grey circle are white; communication in that areas is not affected at all by transmission from the selected nodes.

3 Single-Node Exercises

You will now extend the hello-world example to add more complex stuff. Create a folder in an easy-to-access location. This folder will contain all the projects of this course. Copy+paste the `contiki/examples/hello-world` folder in your new folder. Use an editor that works for you (Eclipse is pre-installed).

3.1 Sensors, LEDs and Button

We will extend the Hello World program to let it print sensor data and toggle the leds when the button is pressed. Use the `hello-world.c` as a starting point. For the button, leds and sensors you need to include some files (see also `examples/sky/test-button.c` from where you can copy code lines:

```
#include "dev/button-sensor.h"
#include "dev/light-sensor.h"
#include "dev/leds.h"
#include <stdio.h>           /* For printf() */
```

Contiki uses protothreads [3] to provide thread-like abstractions on top of a lightweight event-driven kernel. Protothreads are non-preemptive, but provide pseudo-blocking functions to e.g. wait for events, such as `PROCESS_WAIT_EVENT()`. In `hello-world.c`, the main protothread is called `hello_world_process`. The code of this main process is in the function `PROCESS_THREAD(hello_world_process,ev,data)`, between `PROCESS_BEGIN();` and `PROCESS_END();`.
Sensors needs to be activated in the main process:

```
SENSORS_ACTIVATE(button_sensor);
SENSORS_ACTIVATE(light_sensor);
```

To print out the current light sensor value:

```
printf("Light: \%u\n", light_sensor.value(0));
```

To wait for an event and check if this event is a button press:

```
PROCESS_WAIT_EVENT();
if(ev == sensors_event && data == &button_sensor) {
..
}
```

Or, more directly:

```
PROCESS_WAIT_EVENT_UNTIL(ev == sensors_even
  && data == &button_sensor);
```

To toggle LEDs:

```
leds_toggle(LEDS_ALL);
```

Exercise 3. Write a program that loops indefinitely, checks if a button has been pressed, and if so, toggles LEDs and prints out (`printf`) a message. Test it on your mote.

□

3.2 Local Variables

We now extend the Hello World program by adding a counter that counts the number of button press events since bootup.

Exercise 4. Add a **local** counter variable, increment it every time the button is pressed, and print out the current counter in addition to the light sensor data. Test it on your mote.

□

Do you get the expected result? If not, then you have experienced one peculiarity of protothreads: local variables are not stored across a blocking wait. A workaround consists in declaring your variable as `static`. Doing so, the variable is stored in the data segment rather than the runtime stack.

Exercise 5. Fix your counter program accordingly, and test it on your mote.

□

3.3 Timers

We now add a timer to make LEDs blink periodically. Add to your main process:

```
static struct etimer et;
etimer_set(&et, CLOCK_SECOND*4);
```

To check if your timer has expired and reset the timer:

```
if(etimer_expired(&et)) {
  etimer_reset(&et);
}
```

Exercise 6. Write a program that loops indefinitely, waits for an event, then checks if the timer has expired, and if so, toggles the LEDs and outputs a message. Test the program on your mote.

□

Exercise 7. Write a program that loops indefinitely, waits for an event, then checks whether this event is a button press and if the timer has expired. Upon button press, toggle the LEDs and print out "Button press!". When the timer expires, simply toggle the LEDs and print out "Timer!". Test the program on your mote.

□

Exercise 8. Create a Cooja simulation and test your program. You can simulate button press from the menu you get by right-clicking on a node.

□

4 IPv6 Networking

Contiki supports IP networking through the uIP TCP/IP stack. Contiki also has support for IP routing using the RPL protocol that is has been standardised by the IETF (RFC 6550 from RoLL WG). RPL networks are anchored at a root that typically acts as 6LoWPAN border router, i.e. provides low-power mesh network with connection to external networks (Internet, ..).
This part of the tutorial shows how to create Contiki IPv6 applications. We will use the Cooja simulator for most exercises. Most of the code we will use can be found in `contiki/examples/ipv6/` folder.

4.1 Create an IPv6 network

This example will allow to create a simple IPv6 network in the Cooja simulator. We will use an emulated Tmote Sky acting as RPL border router of our IPv6 network. The border router will setup the IPv6 prefix of the network and will initiate the creation of the RPL routing tree. The code of the border-router is in path+contiki/examples/ipv6/rpl-border-router/border-router.c+. Copy and paste the folder in your project folder. You can already copy+paste the whole `examples/ipv6` folder, as we will use most of the code later.
Start Cooja and create a new simulation. Create a new mote type, browse to your project folder where the rpl-border-router is located and compile it. Once the border-router node has been added to the simulation, you then need to populate your network with more motes. For this, we use the code in `ipv6/sky-websense/sky-websense.c`. The sky-websense application generates sensing data and provides access to the latest data via built-in webserver.

Look at the code to see what it does. Add a sky-websense node to your simulation. From the **Network** window, right click on the border-router and select "Mote tools for sky/Serial socket (SERVER)" from the menu. A new window should appear saying that the border-router node is listening on local port 60001. At this point, your simulation is ready to start. Click on the start button, and then do the following in another terminal window:

```
cd to your project folder where the rpl-border-router is located
$ make connect-router-Cooja TARGET=sky
```

This will start a program named `tunslip6` that sets up an interface on the Linux IP stack and connects this interface via a socket to the border router node in Cooja. The Linux interface will be configured so that all IP traffic destined to addresses starting with `aaaa:` will go to the interface and into Cooja.
Note: an IPv6 address is 128 bits long, but addresses can be written more compactly by collapsing zeroes. For example, `aaaa:0000:0000:0000:0212:7401:0001:0101` *is equal to* `aaaa::212:7401:1:101` *in the shorter representation.*
When this program is running there will be some print-outs and there will be lines between some of the nodes in the network visualizer in Cooja. You should see something like

```
ifconfig tun0 inet ‘hostname‘ up
ifconfig tun0 add aaaa::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC  HWaddr 00-00-00
        -00-00-00-00-00-00-00-00-00-00-00-00-00

        ...


*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
 aaaa::212:7401:1:101
 fe80::212:7401:1:101
```

This means that the border-router node has received its global address and has started to construct the routing topology. The IPv6 network is now up and running. Let's test it with some pings from another terminal window (copy the ping6 command into the terminal window):

```
 $ ping6 aaaa::212:7401:1:101
 $ ping6 aaaa::212:7402:2:202
```

If everything works fine, you will get an output that looks like

```
 $ ping6 aaaa::212:7401:1:101
PING aaaa::212:7401:1:101 56 data bytes
64 bytes from aaaa::212:7401:1:101: ttl=64...
time=70.3 ms
64 bytes from aaaa::212:7401:1:101: ttl=64...
time=62.8 ms..
```

After this you should be able to view in your web browser the following IP:

```
http://[aaaa::212:7401:1:101]/
```

And see the following:

```
Neighbors
fe80::c30c:0:0:9e

Routes
aaaa::c30c:0:0:9e/128 (via fe80::c30c:0:0:9e)
```

The actual IPv6 prefix is set in your tunslip6 application Keep adding motes and watch your network grow!
On the sky-websense node, the sensors enabled for this test are temperature and light. You can consult the webserver by searching for `http://[``webserver-ipv6-addr'']` in your web browser to see the available readings.

```
Current readings

Light:  65
Temperature:  24° C
```

Exercise 9. Add several sky-websense motes to your simulation to have multi-hop communication. Try to ping the nodes. Which values of Time to Live (TTL) do you observe? What about the response time? Try to find a node that when pinging will get a TTL of 60. Please explain why the TTL is lower than when pinging the node with the address `aaaa::212:7401:1:101` (the border-router).

□

Exercise 10. Try out the web-server that is running in each of the nodes. Try the following URLs (cut-n-paste them into Firefox):

- `http://[aaaa::212:7401:1:101]`
- `http://[aaaa::212:7402:2:202]`

Now try to access the light service (with `l` for 'light'), and the temperature service (with `t` for 'temperature'):

- `http://[aaaa::212:7403:3:303]/l`

□

4.2 Configuring the Network Stack

Contiki includes a number of radio duty cycling (RDC) layers, such as ContikiMAC, LPP (low power probing), and NullRDC. As for the MAC layer, CMSA and NullMAC (non-persistent CSMA) are supported. The default RDC and MAC layers in Contiki on Tmote Sky are ContikiMAC and CSMA (see `contiki/platform/sky/contiki-conf.h`).
Now let's change the MAC layer to NullMAC. The recommended way to configure the MAC layer is to use a `project-conf.h` file in the directory where your application resides. The project-conf.h file looks as follows to use NullRDC and NullMAC.

```
#ifndef __PROJECT_CONF_H__
#define __PROJECT_CONF_H__

#undef NETSTACK_CONF_MAC
#define NETSTACK_CONF_MAC NullMAC_driver

#undef NETSTACK_CONF_RDC
#define NETSTACK_CONF_RDC NullRDC_driver

#undef CC2420_CONF_AUTOACK
#define CC2420_CONF_AUTOACK              0

#endif /* __PROJECT_CONF_H__ */
```

Finally, you need to ensure that your Makefile in your application's directory declares the project-conf.h file. For this, edit the Makefile and add as first line:

```
DEFINES=PROJECT_CONF_H=\"project-conf.h\"
```

As you just edited your Makefile (with an effect on dependencies), clean your current build:

```
make clean TARGET=sky
```

Check the boot-up output of the node to see if the change has effect and the new MAC layer is used.

Exercise 11. What is the effect of using ContikiMAC+CSMA vs. NullRDC+NullMAC in terms of code size of the firmware?

□

The file `contiki/contiki-default-conf.h` contains a well-documented list of useful system parameters that can be tuned. The Contiki configuration process is quite flexible but also a bit complex:

1. `contiki/contiki-default-conf.h` declares global defaults

2. `contiki/platform/[PLATFORM_NAME]/contiki-conf.h` overrides the global defaults with platform-specific values

3. your own `project-conf.h` overrides both global and platform defaults, setting project-specific values

Exercise 12. Find out how to set the number of routing entries (by looking up `ROUTES` in `contiki/contiki-default-conf.h`). Configure your system to use 8 routes. What is the effect of the number of routes on firmware RAM usage? □

4.3 Measuring Power Consumption

You can also measure the power consumption, e.g. to get an idea of the profile of the different MAC layers or for a scientific evaluation in your research paper. Contiki has a software-based power profiler [2] that basically measures the times different components are turned on. This time can be multiplied with the voltage and a pre-measured current draw which approximates power consumption. Using this mechanism requires only some extra lines of code. Here we do this for the RX time, i.e., the time the radio is receiving or performs idle listening.

```
#include "energest.h"

...

static unsigned long rx_start_duration;

..

rx_start_duration = energest_type_time(ENERGEST_TYPE_LISTEN);

// in a while loop
while(1) {
...
    printf("energy rx: %lu\n",
energest_type_time(ENERGEST_TYPE_LISTEN) - rx_start_duration);
...
  }
...
```

This gives you the time the radio was in RX mode (`rxon`). You also need the total time that has passed, which you usually compute by summing time spent by the cpu in active mode (`ENERGEST_TYPE_CPU`) and low-power mode (`ENERGEST_TYPE_LPM`). An approximation of the power consumption from Radio RX can be obtained as follows:

$Power(mW) = \frac{rxon}{cpu+lpm} \times 20mA \times 3V$
The 20mA are pre-measured (or from data sheet). 3V is the Tmote operational voltage (approximated).

4.4 Turning the radio on and off

You can easily turn the radio off and on from the application, e.g., to save energy when no communication is expected.
This can be done by doing, e.g.:

```
#include "netstack.h"

if (seconds % 2 == 0)
   NETSTACK_MAC.on();
else
   NETSTACK_MAC.off(0);
```

Exercise 13. Modify your `sky-websense.c` application so that it prints the energy consumption every 4 seconds. Then, add a `project-conf.h` configuration file to your application and try several RDC layers (ContikiMAC, X-MAC, LPP, NullRDC). Do you see any changes in the energy consumption?

□

Exercise 14. Modify your `sky-websense.c` application so that it turns off the radio every second. Measure the RX power consumption every second to see the effect. It might be easier to use NullMAC if you want to verify your measurements are correct (an always turned on cc2420 radio consumes around 60 mW).

□

4.5 REST/CoAP exercise

In this exercise, you will setup an IPv6 network with CoAP resources. A CoAP client (your web browser with the Copper plugin) polls the CoAP resource on the mote, and displays its content. We will do the exercise in Cooja first.

4.5.1 Preliminaries

First, get the Copper (Cu) CoAP user-agent from `https://addons.mozilla.org/en-US/firefox/addon/copper-270430`. Copper is a CoAP plugin for Firefox that allows you to access a resource via CoAP.

4.5.2 Setup in Cooja

In Cooja, open the simulation `contiki/examples/er-rest-example/server-only.csc` You can copy+paste the `er-rest-example` folder in your project folder. Take the border-router from our previous exercises.

Open a new terminal and launch the tunslip6 application:

```
cd contiki/examples/er-rest-example
make connect-router-cooja
```

Now you can start discovering the resources. Open Firefox (Cooper should start automatically) and type the server address:

```
coap://Cooja2:5683/
```

The address 'Cooja2' may be substituted with the IPv6 address, e.g., `[aaaa::212:7402:2:202]` To start discovering the available resources, press "DISCOVER" and the page will be populated in the left side. Then, you can try some of the available features

- select the `toggle` resource and use POST to see how the RED led of the server mote will toggle
- choose "Click button on Sky 2" from the context menu of mote 2 (server) after requesting `/test/separate`
- do the same when observing `/test/event`
- select the `Hello` resource, the server will answer you back with a well-known message
- if you observe the `Sensors-Button` events by selecting it and clicking "OBSERVE", each time you press the user button an event will be triggered and reported back

4.5.3 Setup on Tmote sky

Now, you can setup your CoAP-based network using real motes!
For this, you need a Tmote Sky mote attached to the USB port of your laptop running the border-router process. A second mote should run the CoAP server. You can refer to the `README` file in `/examples/er-rest-example/` for more instructions. Try to ping the CoAP server from your computer and access its CoAP resources from your web browser.

References

[1] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In Workshop on Embedded Networked Sensors, Tampa, Florida, USA, nov 2004.

[2] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In Proceedings of the Fourth Workshop on Embedded Networked Sensors (Emnets IV), Cork, Ireland, June 2007.

[3] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006), Boulder, Colorado, USA, November 2006.